Is It Possible to Legislate Fair Share?

Abstract

Lost in the debate surrounding net neutrality is the assumption that all end hosts play nice and "fair share" the limited network bandwidth. Traditionally, "fair share" is enforced through a combination of forces: there is a general consensus on the TCP congestion control algorithm, operating system vendors implement such algorithm in the kernels and it cannot be changed, applications use TCP and use only a small number of TCP connections, and perhaps most importantly, network operators can identify abusive applications and traffic-shape them.

Unfortunately, today these protective forces are being weakened. The networking research community has proposed at least nine different congestion control algorithms, some more aggressive than others. Popular kernels such as Linux have implemented all of the algorithms, and furthermore, allow users with proper privileges to modify the congestion control algorithm arbitrarily via loadable kernel modules. Many P2P applications use four or more connections between each host pair to improve their performance. And finally, most proposals of Net-Neutrality legislation prohibit network operators from traffic-shaping applications, for good reasons.

Therefore, for net-neutrality laws to avoid causing congestion collapse on the Internet, the law must also define proper end-host behavior, i.e. "fair-share", and allow network operators to detect violations of fair-share and apply punishment. This presents three immediate difficulties. First, TCP only provides per-flow fairness, so to achieve fair-share, one application shouldn't use more flows than the other. But having a law to limit the number of TCP connections that an application can open is draconian to say the least. Second, the TCP congestion control algorithm is the key in providing per-flow fair-share, but the algorithm itself is under active research, and legislation in this area could stymie innovation. Finally, it's difficult to have cheap and yet accurate detection of fair-share violations in routers, and any approximation techniques risk penalizing the innocent.

This paper sketches out a set of potential solutions out of this dilemma. A central theme of the proposal is that there should be a code of conduct for "fair-share" involving the number of TCP connection used and the TCP congestion control algorithms used, and there should be a differentiation between public hosts and private hosts. Public hosts pledge to follow the "fair-share" code of conduct, and allow the behavior of their traffic to be monitored and their pledges verified. In return, they receive a guarantee from network providers that their traffic always receives the fair-share and won't be shaped. Private hosts do not have to be constrained by the "fair-share" code of conduct, and in return they accept that their traffic will be shaped from time to time as deemed appropriate by the network provider. Similarly, there is a code of conduct for applications that utilize private hosts' networking resources for community benefit, and a separation between "public" applications that won't be traffic-shaped and "private" applications that can be.

1. Notes to the Program Committee

I apologize for a hastily and poorly written paper. I am somewhat embarrassed to submit such a poorly written paper in the first place. However, the issue raised here is important; I haven't seen a good discussion about it anywhere. By submitting the paper, I am hoping to make this a discussion topic at HotNet. I'll definitely refine the paper should you decide to accept it in some form.

This is an anonymous submission. Regardless of the anonymity, the opinions and views expressed in this paper in no way, shape or form reflect those of the author's employer.

2. Introduction

TCP is the protocol used by virtual all end hosts for reliable communication and is a cornerstone for Internet stability. The congestion control mechanism in TCP ensures that data flows traveling along a network link fair-share the link bandwidth, and when congestion occurs, the sending rates of the flows are reduced. Details for TCP can be found in many text books.

The fair-share property of TCP congestion control is derived from its AIMD (additive-increase multiplicativedecrease) algorithm. In TCP, the sender's sending rate is controlled by the congestion window, W. The flow cannot have more than W packets in flight. When a packet loss is detected, W is halved. Afterward, W is only increased by 1 per round trip in order to probe for spare bandwidth.

The AIMD algorithm is proved to be fair. When the RTTs of the flows are the same, even if one flow starts with a much bigger share of the network bandwidth than another flow, they converge to fair share of the network bandwidth. This has been proved in a theoretical model, and have been validated by simulations. The actual implementation of TCP has went through multiple iterations, with Reno as the basis and SACK and FACK added later on.

However, AIMD only provides per-flow fairness. If application A uses 2 TCP flows while application B uses only 1 TCP flows, and both A and B travel the same bottleneck link, then A will receive twice the share of the bandwidth. Indeed, many P2P file sharing application uses many TCP connections between peers and, left unconstrained, would monopolize the Internet bandwidth, as shown in [2].

So far, the end-to-end principle of Internet design has worked reasonably well due to the following reasons:

- there is a clear consensus on the TCP congestion control algorithm, i.e. AIMD.
- AIMD is implemented inside popular operating systems, and the applications cannot change its behavior.
- Popular applications use just 1 or 2 connections between hosts.
- Network providers have the right to traffic-shape data traffic that are blatantly exploiting network resources.

These are protective forces that enable the Internet to function. (We are ignoring the discussions of UDP here, since major applications on the Internet today are no longer using UDP, and the Slammer worm has made rate-limiting UDP a necessity that no one is challenging).

However, today these protections are being weakened:

- *Loss of the consensus*: Due to AIMD's poor performance in long-distance fat-pipe networks, TCP's congestion control algorithm has been a very active research area. There are no less than nine proposals floating around. Some of these claimed to be TCP-friendly. Some have already been shown to be quite unfriendly to TCP Reno. Nevertheless, they have been all implemented in Linux.
- *Ease of modification of TCP*: It's no surprise that, with so many algorithms on the table, the operating system developers would decide to make the TCP implementation modular, and in the process, make them kernel modules that can be changed by any user allowed to load up kernel modules. Starting in Linux 2.6, Linux has introduced the TCP congestion control API, which made several key components of TCP stack operations hooks that a user can register modules with. If a user wanted to, he or she can make the TCP run arbitrary congestion control functions.
- *Loss of Net-Etiquette in Programmers*: It used to be that distributed applications were developed by programmers that are sensitive to fairness in networking. Rise of P2P file-sharing applications changed all that.
- *Threat of Legislation*: Most proposed Net-Neutrality legislation outlaws traffic shaping, for very good reasons: traffic-shaping can be easily abused to discriminate certain applications.

If we were to enact laws prohibiting network providers from bandwidth-limiting data traffic, then we should also enact laws prohibiting end-hosts from unfair behavior. But is it possible to define what behaviors are fair and what are unfair?

Any definition of fair-share has to include both limits on the number of TCP connections an end-host/application can use and and requirements on the congestion control algorithm used. However, it is simply unfathomable for a law to decree how many TCP connections an application can use. Furthermore, given the difficulty that AIMD experiences in large Bandwidth-Delay-Product (BDP) networks, it'd be difficult to legislate the congestion control algorithm too.

To resolve this difficulty, we propose that the hosts on the Internet be put into two categories, public or private, with different constraints and privileges accorded to both. Similarly, we propose that applications be put into two categories, public or private. We elaborate below.

3. Public vs Private Hosts and Applications

After years of security attacks, the Internet has evolved into a cluster of walled fortresses called Intranets. Practically every business employs firewalls to prevent access to internal machines from the Internet, and most, if not all, residential users have firewalls or gateways to protect machines at home. In fact, researchers have proposed that the Internet be a "off by default" network [1].

The types of hosts that are "open" on the Internet are quite limited, mostly web sites and DNS servers. These hosts run a very limited set of well-defined applications. These hosts, or rather, businesses associated with these hosts, care that the traffic sent by them are not rate-limited by network service providers.

A compromise can thus be made that such hosts (actually, the IP addresses of these hosts) be declared "public" hosts, and constraints be put on these hosts in return for the privilege of not being shaped. All other hosts are considered "private" hosts. No constraints are put on their behavior, including what kernels they use, what applications they run, and how many TCP connections they use. In return, their traffic will be subject to shaping when deemed appropriate by their network service provider.

Similarly, applications can declare whether they fair-share or not. Applications such as P2P systems that utilize private hosts for community benefits have a right to see that their traffic are not limited by network providers, but to gain that right they need to promise that they do not unfairly take resources away from other hosts and other applications. Since many Internet service providers are built out with an assumption of the traffic mix from their user base, the code of conduct of "fair-share" for applications inevitably involves promises of concurrency of peer activities.

4. Code of Conduct for Hosts

We propose the following fair-share code of conduct for public hosts:

- Public hosts do not initiate TCP connections to other hosts. Instead, they behave as mere receivers.
- The client agents that connect to the public hosts should use only up to 2 TCP connections for data, and 1 TCP connection for control.
- Public hosts use BDP-appropriate congestion control algorithm. That is, for BDP below certain range, TCP Reno AIMD congestion control is used. For BDP above certain range, a small set of TCP congestion control algorithms, to be determined by the research community, are allowed.

Monitoring and verification of the code of conduct can be performed by any network service providers that carry traffic between private hosts and public hosts. The direction of the connection is easy to monitor.

The number of connections are harder to monitor. In the case of public hosts having no control over the client agents that run on private hosts to connect to them (for example, web sites and browser applications), there is little chance for public hosts to cheat. In the case of public hosts having client agents installed on private hosts to connect to them, the service provider best situated to verify that those agents are not cheating is the service provider directly connected to the private hosts, i.e. "eyeball ISPs".

Verifying that the public hosts are using the claimed congestion control algorithm is perhaps the hardest. Here, we propose two approaches, one macro-level monitoring, and one micro-level random testing.

Macro-level Monitoring The macro-level monitoring approach is best performed by the "eyeball ISPs", which also has the most interest in making sure that the public hosts do not cheat. Its principle is very similar to the TCP-friendly test as advocated in [4].

In the macro-level monitoring approach, the router estimates, for a public host under monitoring, the aggregate drop rate for flows from the host (p), the total number of flows (K), and the RTTs between the private hosts and the public host (R).

Given the three estimates, assuming that the aggregate drop rate p reflects the average drop rate of each flow, the standard TCP friendly test [4] says that the total packets sent by the public host should be bounded by $K * (1.5 * sqrt(2/3))/(R * sqrt(p))$. Note that any drop rate observed by one router is a lower bound on the actual drop rate, thus the sending rate is upper-bounded by the formula here.

Estimating the aggregate drop rate is easy: just keep the total number of dropped packets that are from the public host and the total number of packets sent by the public host.

The total number of flows to the public hosts can be estimated by using the probabilistic counting approach [3]. Probabilistic counting uses very little state and can be implemented efficiently. With a few counters, the estimate can be made very close to the actual count.

The RTT between the public host and the private hosts served by the router can also be estimated relatively easily, through a variety of existing methods.

Micro-level Random Testing A router can randomly test if a particular TCP flow follows a particular congestion control algorithm by selecting a flow and emulating the TCP stack from the connection establishment to tear-down. Even though the router might not see the packet drops from the flow, it can infer the packet drops from retransmitted packets.

5. Code of Conduct for Applications

For better or worse, the current ISPs are built out with certain assumptions about the statistical multiplexing of the users' traffic. By assuming that not all users will be sending or receiving data at the same time, the cost of the network is reduced, and the reduced cost *should have* translated into lowered costs for consumers.

Current P2P applications, unfortunately, completely violate these multiplexing assumptions. They take network resources away from other hosts. This is the reason why ISPs are forced to rate-limit these applications.

Businesses using P2P applications for community benefits have a legitimate demand that the applications' traffic not be rate-limited. On the other hand, the ISPs also have a legitimate demand that these P2P applications do not monopolize network resources at the expense of others.

To escape this dilemma, we again propose a fair-share code of conduct for applications that utilize private hosts. The code of conduct reflects the following:

- the P2P application can only use at most 2 TCP connections between private hosts;
- the P2P application should make its TCP flow clearly recognizable by routers, either by running on fixed ports, or by having a clear identifier in the first packet sent after connection establishment, so that the deep packet-inspection engine at the router can identify the flows and appropriate tag them;
- the P2P application agrees to abide by the multiplexing constraints of any ISP whose private hosts are peers in the application. That is, if an ISP demands that only C hosts can perform data transfers for the application at the same time, then the P2P application should abide by that demand.

The last requirement means that the routers need to communicate with the P2P applications. To signal its constraints to the P2P application, the router can use two approachs. One is to simply deny connections between two peers if the number of connections between peers in a P2P application exceeds a limit. This gives a clear signal to the P2P application; however, it does not suite the need of applications that keep connections with lots of peers but most of the connections are idle.

The second approach is for a router to apply congestion signal collectively. That is, when dropping one packet from a flow belonging to the P2P application, the router drops packets from other flows belonging to the same application, so that more flows from the P2P applications back off. (Since the concern in this case is not that private hosts manipulate their own kernels to help the P2P application, one can assume that the private hosts will use proper TCP congestion control algorithm and the flows will back off when experiencing loss.)

Assume that the router detects n active flows belonging to the P2P application and the multiplexing assumption says only m should be active, then whenever the router drops one packet from a flow, it drops one packet from $(n/m)^2$ other flows from the same application. Since the flows are clearly recognizable, this can be easily done.

In a way, the second approach is a way of rate-limiting. However, unlike the current traffic-shaping practices, this form of rate-limiting is explicit, and the P2P applications can adjust to it.

6. Discussions

It has been noted at a recent NANOG meeting that, while content providers fear that they might be discriminated by the ISPs, ISPs are thinking about BitTorrent when they vehemently oppose Net-Neutrality, There is a clear disconnect.

Both proponents and opponents of Net-Neutrality make good arguments. Perhaps, by clearly acknowledging the role that end hosts and applications play in affecting distribution of network resources, we can create a common ground for discussion.

References

[1] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by default! *Proceedings of the ACM SIGCOMM HotNets-IV*, 2005.

[2] K. Cho, K. Fukuda, H. Esaki, and A. Kato. The impact and implications of the growth in residential user-to-user traffic. *ACM SIGCOMM Computer Communication Review*, pages 207–218, 2006.

[3] P. Flajolet and G. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 1985.

[4] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.